SMPTE
Society of Motion Picture and Television Engineers

595 West Hartsdale Ave.
White Plains, NY 10607
(914) 761-1100

RECEIVED
FEB 20 1992
Federal Communications Commission
Office of the Secretary

EX PARTE OR LATE FILED

ORIGINAL FILE
87-268

For immediate release
Contact: Carol King

SMPTE Approves Task Force Report on Headers/Descriptors

White Plains, N.Y., February 7, 1992 -- The Report of the Task Force on Headers/Descriptors was approved unanimously at the full meeting of the SMPTE Standards Committee meeting held on Feb. 6, 1992 in San Francisco. The Committee expressed its appreciation to the Chairman of the Task Force, Dr. Will Stackhouse, Jet Propulsion Laboratories, and to the Vice-Chairman, David Staelin, Massachusetts Institute of Technology, for their leadership in completing the complex report in the remarkably short time of seven months and to the members of the Task Force for their expertise, insight, and close collaboration in the preparation of its valuable contents.

The report, in essence a feasibility study of possible methods to implement a header/descriptor mechanism, has been referred to the SMPTE Committee on Television Production Technology for further action, including the necessary work leading to detailed SMPTE documentation of the format, construction, and usage of the header and descriptor for the interchange of image, sound, and related data between systems. It is anticipated that the high level of collaboration that has existed between the representatives of various industry segments will continue into this work and lead to a rapid convergence of views concerning the contents of the documents.

In essence, the proposed header is a digital label, identifying the encoding standard and the size of the data block contained in the associated envelope. It may also indicate the presence of a readable descriptor. The header is the enabling mechanism for the flexible exchange of picture, sound, or other data between diverse systems, providing the necessary unambiguous information for the identification of the associated data. The design of the header must consider the following attributes:

Universal. All relevant data blocks must be labeled and identified to enable unambiguous interpretation.

Interoperable. The header enables the sharing of data streams across diverse applications, equipments and environments.

Extensible. Service enhancements and technical innovations can be accommodated without obsolescence.

Cost Effective. A range of equipment having differing cost/performance characteristics can be accommodated within a system.

No. of Copies rec'd 2
List A B C D E

(more)

Economic. The header occupies a very small percentage of the data stream (several orders of magnitude less than simple sync in NTSC).

Efficient. The header data can be rapidly acquired after changes in source or content of data streams, thus allowing editing, switching and transmission of differing formats in a system.

The descriptor is a block of data that enhances the utility of the main data for the user. It may contain, in standardized format, data concerning production, ownership, access, previous processing, or other information additional to the basic interpretation of the data. In simple processes, the descriptor may be skipped.

The header/descriptor is the key to the efficient and flexible use of the digital data stream for the communication, storage or display of digitally-expressed pictures, sound, text or other items and makes possible scalable, extensible systems. It serves to identify the specific attributes of a data service between processes and thus enables the interoperability of systems using differing, but predetermined, standards.

The SMPTE is now undertaking the documentation of the standard for a header/descriptor that will apply to television, multi-media, image transfer, and a wide range of other related applications. It anticipates a close liaison with other groups involved in, or affected by, this work and is actively seeking their participation to ensure standards having the widest use and maximum economy of application. The work represents a major and practical step towards the goal of fully flexible, interoperable, scalable, and extensible systems that so many are seeking. Television, HDTV, HRI, graphics, and image communications will at last be able to overcome many of the barriers to the free flow of material.

The completion of the preliminary work on Headers/Descriptors, enables the Task Force on Digital Image Architecture, chaired by David Trzcinski, PictureTel, to proceed rapidly with its assigned work, building on this base, and it is expected to complete its Final Report during the middle of 1992. Standards derived from this work by SMPTE, or by other organizations, will notably facilitate the flow of images between systems, especially across differing applications or industries.

Society of Motion Picture and Television Engineers®
595 WEST HARTSDALE AVENUE, WHITE PLAINS, NY 10607-1824
TELEPHONE: (914) 761-1100 / TELEX: 4995348 / FAX: (914) 761-3115

This Report of the SMPTE Task Force on Headers/Descriptors is an approved document of the SMPTE Standards Committee and is made available for information, as it contains valuable proposals concerning the development of digital imaging and video systems and for standardization of certain of their aspects, that will be of interest generally.

The standardization aspects of the Report will be further considered under the normal processes of the SMPTE for the creation and approval of Engineering Documents, which includes the opportunity for further comment and for public review prior to their final acceptance. Persons wishing to actively participate in the development of these standards, including attendance at Working Groups meetings and ballot response, may contact the Engineering Department of the SMPTE at the above address.

It should be noted that Engineering Documents arising from the contents of this Report may differ significantly from its recommendations and caution is suggested in the use of this report as the basis of design or of implementation.

SMPTE HEADER/DESCRIPTOR TASK FORCE

FINAL REPORT

January 3, 1992

1.0 INTRODUCTION

The Task Force on Header/Descriptors has considered the questions posed in its Scope of Committee Work, and makes the following final report and recommendations to the Standards Committee.

The report begins with a discussion of the general objectives of the header/descriptor, and then presents more specific objectives selected by the Task Force as it developed two alternative implementations.

Both of the proposed implementations could support new SMPTE standards, and are described in some detail here. The "ASN.1 Implementation" is structured using only Abstract Syntax Notation 1 (ASN.1), an existing and evolving ISO/CCITT standard principally used in the computer industry. The "Compact Implementation" is designed to minimize the number of bits allocated to the header/descriptor function, but also permits optional use of the ASN.1 notation later in the header/descriptor for further extensibility. Both implementations perform essentially identical functions.

Appendices A and B present illustrative approaches to the design of transport headers and header-decoding software, respectively. Transport headers are designed to address certain difficult data transport problems. Appendix C lists the official task force members as of January 3, 1992.

In view of 1) the great importance to industry and its customers of the capabilities provided by the header/descriptors described below, and 2) the degree to which these two possible implementations satisfy the objectives established at the outset for header/descriptors, the Task Force recommends that:

> The Standards Committee arrange for the preparation of one or two new standards for digital header/descriptors based on either the "Compact" or the "ASN.1" Implementations described below, or on a combination thereof.

2.0 GENERAL OBJECTIVES

The header/descriptor task force was directed to consider header/descriptor architectures and implementations appropriate for the emerging digital high-definition television (HDTV) and high-resolution system (HRS) industries. The primary design objectives of the task force are:

- Universality -- All image and other data streams should be labeled so that signals can be shared across systems and applications with minimal degradation or confusion; the header/descriptor should therefore uniquely identify the encoding scheme employed and how the data is to be interpreted.

- Longevity -- The header/descriptor should provide a number of potential identification codes adequate to serve for decades, and preferably centuries; this implies that specific encoding identifiers, once assigned and registered, should not be reassigned or redefined. The header/descriptor should also facilitate longevity for equipment and media of all types.

- Extensibility -- To facilitate service enhancement and innovation, and to promote longevity of both equipment and recorded signals, the header/descriptor should accommodate technological advances in either equipment or recorded signals with minimal risk of obsoleting existing components, infrastructure, and media collections.

- Interoperability -- The header should permit optimal sharing of data streams across data-generation, carrier, and equipment technologies and services in a variety of error environments, and should permit all equipment and applications to successfully ignore encrypted or otherwise deliberately inaccessible data.

- Cost/Performance Effectiveness -- The header/descriptor should permit use of both low-cost equipment as well as more expensive high-performance equipment; the header/descriptor should also accommodate inexpensive equipment incapable of decoding all possible data streams. Economy and simplicity through flexibility and scalability of the key performance parameters should also be supportable.

- Compactness -- The header/descriptor should be economic in its utilization of bits, and should typically comprise a negligible fraction of the underlying data stream.

- Rapid Capture -- Much video and other serial data is intercepted mid-stream, such as when users switch to a new channel, and therefore the header/descriptor should permit rapid header identification, adequate to meet the needs of all applications.

- Editability -- Common editing and parsing operations, such as splicing, appending, replacing, inserting, cropping, and overlays, should be supportable by the header/descriptor architecture without necessarily requiring decoding and encoding of the data stream itself.

3.0 SPECIFIC HEADER/DESCRIPTOR OBJECTIVES

To meet the general objectives summarized above, the Task Force selected the following compact set of specific objectives which are met by both implementations described later. The header and descriptor are defined here separately.

3.1 Specific Header Objectives

The specific objectives of the header are to:

- Identify by number the encoding standard employed by the attached block of data.
- Specify the length of that block of data, so that equipment of any epoch can successfully skip uninteresting blocks of data or data encoded using standards defined subsequently.
- Indicate whether a readable descriptor follows the header.
- Permit users to intercept data streams at random times, as when switching channels, so that proper data interpretation begins swiftly.
- Provide optional error-protection capability. Data generation entities may wish to supplement error-protection services provided in subsequent environments experienced by that data, particularly when those environments are unknown.

The task force considers these attributes of the header to be the minimum mandatory set, recognizing that additional important capabilities can be provided by the descriptor.

3.2 Examples of Header Use

A simple example illustrates how these minimal capabilities for the header satisfy the general objectives discussed above. Suppose, after many years, some HDTV broadcasters wish to provide dual-language sound tracks. This capability could be provided by adding to the data stream blocks of data conveying the second language. These new blocks would be labeled by a header incorporating a standard number not recognized by equipment produced earlier. This older equipment would read the header and recognize the standard identification number as being unknown. It could then observe the length of the associated block, and skip over it to the next header.

All data could be labeled by such flexible headers, or only a designated portion (e.g. "auxiliary data") of a more rigidly defined larger video data stream. Note that HDTV receivers capable of receiving only 20 Mbits per second could not accommodate increases except at the expense of any spare capacity previously reserved for expansion, or by the broadcaster reducing the number of bits conveying video or audio; in the latter case the original standard would have to be defined so as to permit receivers to accommodate any such real-time video or audio truncation, however.

3.3 Specific Descriptor Objectives

The principal function of the descriptor is to convey additional information that improves the usefulness of the data to the user; its format would be specified independently of the standard employed for the data itself. Such optional auxiliary information in the descriptor might include transport information such as cryptographic, priority, or additional error-protection information, as well as source time, authorship, ownership, restrictions on use, royalty payment information, explicit description of encoding or decoding processes, intermediate processing performed, and other information in forms that could evolve over the years. To simplify the decoding task, the descriptor may also contain an abbreviated "table of contents" and a flag indicating whether any information has changed since the previous descriptor. The beginning of the descriptor would also indicate the descriptor length so that it might be skipped without interpretation if the user chooses. Optional additional error protection would be available for data originators so desiring it.

Specifically, the descriptor could include:

- A list of standard-identification numbers, parameters of operation, text, and algorithms, in any desired combination.
- A compact optional table of contents for the descriptor.
- A flag indicating whether changes occurred since the previous descriptor.
- The length of the descriptor so that it might be readily skipped if desired.
- Information indicating the number of descriptor entries and their formats so that they might be properly interpreted.
- Optional error protection for the descriptor.

The presence or absence of a descriptor could be indicated by one of the bits contained in the header.

3.4 Examples of Descriptor Use

The use of standard identification numbers in the descriptor permits very compact and flexible encoding. For example, one such number might be allocated internationally to each model number of studio television camera, so that subsequent image processing can maximally improve image quality, compensating for any camera idiosyncrasies. Similar identifiers could be used for different forms of physical, analog, or digital filtering that has been applied to the image subsequently, so that user equipment might again appropriately refilter the image in an optimum way depending on the user's intentions. This is important because performance for any particular display device or audio system is best when that processing reflects the processing that has occurred previously.

Authorship, ownership, and other such information could be conveyed by compact standard-identification numbers, or by use of plain text in English or some other language. Certain descriptors might simply be numbers indicating the settings of certain switches at the time the signals were generated, such as switches controlling audio base, treble, or volume. The descriptor may also include subroutines or other encoded instructions that facilitate subsequent processing or decoding.

Standards numbers and parameter fields can also be used to support transport-layer functions, including essentially all forms of cryptography, statement of the relative priority of the current data block, priority "bidding" data (so users can bid for priority in a free-market sense), synchronization reinforcement blocks, and other information, the character of which can be defined over the years as new standards-identification numbers are assigned and as new languages and protocols are defined. The incorporation of transport capabilities in this standard should not compromise those established by other layers, but would merely supplement them. The only limitation is that such descriptor transport standards should remain robust if block sequences are shuffled in another transport layer.

4.0 ILLUSTRATIVE EXAMPLES OF HEADER/DESCRIPTOR USE

An HDTV broadcaster could simply divide the HDTV signal into blocks, each beginning with a header of perhaps 6-16 bytes' length. This header would contain the length of each block, which could be fixed for all time or variable, and a unique standards number indicating the given HDTV encoding protocol, which may also be unchanging in the initial years. If the over-the-air broadcast standard is heavily error protected, little additional error protection might be added to the header. Good engineering practice would suggest, however, that the header be independently error protected using some of the options described later, and that separate, and possibly less robust, error protection be applied to the remainder of the data stream.

If the HDTV channel is defined so as to perform all transport functions, including all synchronization and error correction, then the header/descriptor described here might be imbedded in the transported data stream. At that level it would preferably be used to encapsulate all data, but could be used in an inferior implementation to encapsulate and characterize only substream or side-channel data. In this example too, the descriptor might convey the origins and processing history of the data, enabling future higher-performance systems to employ post-filters optimizing the quality of the output images. Such flexibility could be particularly important if the output display capabilities of the equipment enabled flexibility in frame rate, pixel interpolation routines, and chrominance manipulation. In the case of audio signals, descriptors could be used in a similar manner to enable optimum reproduction by characterizing microphone placement and preprocessing.

A variety of header/descriptors plus associated data could be sequenced in any side channel to provide a variety of flexible delivered products, such as multiple audio channels, multiple-language captioning, home-shopping order information, and even entire TV signals encoded at lower quality so that more than one can share a single channel. In this example, the ultimate flexibility that could be obtained with individual broadcast channels would be heavily dependent upon the flexibility inherent in the broadcaster's ability to decrease the data rate associated with the initial broadcast product to accommodate possible growth in use of the side-channel capability.

It is also possible to send descriptor information in a separate block with its own header, rather than imbedding it in the same blocks as the associated data. In this case the characterization of a signal provided by a descriptor could remain unchanged for many blocks until that descriptor information changes. In the event the transport layer is prone to shuffling the sequence of blocks, impairing association of descriptors and use of this data, the inherent flexibility of this header/descriptor system permits incorporation of sequence numbers in blocks so that the receiving entity can properly sequence them. Such separation of descriptor information into separate blocks also simplifies translation between environments having different transport layer protocols; any descriptor elements providing transport-layer functionality can be added or deleted when moving between such environments, as desired. An option whereby special "transport header/descriptors" are prepended to blocks is described in Section 5.7.

To the extent that transport-layer functionality might be imbedded within the basic data block, it could be harmlessly overlaid by similar transport-layer functionality in other system elements without loss. Thus the great flexibility inherent in the header/descriptor architecture proposed here, including its ability to perform multiple functions in a multi-layer ISO environment, should not be a handicap, and may in some applications be an important advantage.

Furthermore, this header/descriptor structure permits efficient utilization of the standards activities of a wide variety of national and international organizations. Every standard developed by such bodies for characterizing or communicating digital information can be characterized by an identification code which can be conveyed in an efficient manner by the header/descriptor system described here. Since the implementations proposed here call for each standards authority to have its own identification number, such standards bodies could, in this "ID" concept, choose to use the very same identification numbers they have previously chosen for other purposes. Alternatively, that authority could choose some other simple one-to-one mapping between numbers. At the same time, such standards can also use this header/descriptor system as an imbedded construct within their own protocols. Although the full power of the flexible extensible structure proposed here will only become apparent as it is developed and improved over the years, the basic architecture can be fixed immediately. This would permit immediate fabrication and utilization of equipment based upon this standard.

5.0 COMPACT HEADER/DESCRIPTOR APPROACH

5.1 Compact Header/Descriptor Architecture

The header is divided into two parts: a two-byte "header key", and the remainder, or "header tail". Among the available header options are those without tails, corresponding to a two-byte header conveying simple messages; thirty-two possible messages of this type are available for future definition. The most usual function of the key, however, is merely to "unlock" the tail by providing information about the format of that tail.

The descriptor is divided into three parts: a two-byte "descriptor key" (similar to the header key), a "core" of 0-8 bytes, and the "descriptor tail", which conveys a series of numbers signifying various pieces of information.

5.2 Header Key

5.2.1 Header Key Organization

The header key consists of two four-bit fields and one eight-bit field. The eight-bit field provides two-bit error correction capability for the key, and the first four-bit length-type "LT" field determines the length of the header field in the tail which contains the length of the block. A block of data is defined as comprising the header, any descriptors, and the associated data. The other four-bit "ID" field normally determines the number of bits in the header-tail field devoted to indicating the standard number under which the remainder of the data in the block, possibly including the descriptor, is encoded. Most descriptors would be publicly readable, however. The ID field also indicates whether a readable descriptor follows the header; sometimes one might wish to read the descriptor before deciding whether to decode the rest of the data block. The ID field can alternatively convey messages for certain four-bit combinations in the first four-bit "LT" field. In addition to specifying the length of the header tail field devoted to specifying block length, the LT field also determines what level of error protection is being provided for the header.

5.2.2 Header Key: Four-Bit "Length-Type" LT Field

The primary purpose of the 4-bit LT field is to specify the length of the field in the header tail devoted to specifying the block length. For six of the sixteen possible combinations proposed here (the "fixed-length" options) the total length of the block is pre-specified so that no bits in the header tail are allocated to specifying block length. The other ten proposed combinations permit use of one, two, four, and six bytes for specifying the block length in bytes in integer format; in each case versions with and without error protection capability for the header are available to the standards definition community. The eight-bit protection provided to the header key is always present, however, since the integrity of the key is crucial, and it represents such a small part of the total.

The six proposed fixed-length options are as follows:

1. The block is two bytes long and the message is conveyed by the

four-bit ID field; sixteen messages are possible. One of these messages could signify that the rest of the header/descriptor is coded in ASN.1 (dual compact- and ASN.1-header decoding capability would be necessary for all equipment, however, but Appendix B suggests this might not be burdensome).

2. Same as (1), except that an additional sixteen messages are possible (for a total of 32 two-byte options).

3. The block is four bytes long, the message consisting of the four ID bits plus two bytes, a total of twenty bits.

4. The block length is six bytes; twenty-eight bits (4+3x8) are available, the remaining byte providing error protection for the last 4-byte set.

5. The block length is six bytes with thirty-six bits of information (4+4x8) being available, but without additional error protection.

6. The length of the block is unknown or irrelevant.

In each of the foregoing options, except the first two, the available bits can be divided in a yet-to-be-determined way between those indicating the standard identification number and any additional message. The proposed SMPTE standard would constrain only options 4 and 5, allocating the first eight bits to designating the sovereign state, so that development of these six-byte options can proceed without international agreement. Designation of standards bodies and sovereign states is discussed further in Section 5.2.4.3.

The ten remaining proposed options for the LT field provide for either one, two, four, or six bytes in the header tail to be allocated to specifying the block length in bytes, always in integer format. Depending on which of these ten options is chosen, additional information is also conveyed concerning the level of error protection for the header. These proposed options are as follows:

7. One-byte field in the header tail specifies block length in integer format; no additional error correction capability is provided to the header. Blocks up to 256 bytes long are available under this option.

8. Same as (7), except an additional one-byte is provided for header error protection.

9. Same as (7), except that two bytes specify block length; the maximum block length here is 64 KB.

10. Same as (9) with an additional byte for header error protection.

11. Same as (7), except that four bytes specify block length, which can approach four billion bytes.

12. Same as (11), except that two bytes of header error protection are added.

13. Same as (7), except that six bytes specify block length.

14. Same as (13), except that two bytes of header error protection are added.

15-16. To be determined.

5.2.3 Header Key: Four-Bit ID Field

When the 4-bit ID field is not being used to convey a message using LT options 1-6, then the first three bits of the ID field indicate one of eight possible lengths for the header-tail field conveying the standard identification number, and the fourth ID bit indicates whether a readable descriptor follows the header. These proposed eight tail-length field options include one-, two-, four-, and eight-byte options. These eight tail-length options proposed for the ID field are as follows:

1. One-byte standard identification number allocated internationally.

2. Same as (1), but providing for 256 additional possible standards.

3. Similar to (1), except that the standard field in the header tail contains two bytes instead of one, providing for 64,000 international standards.

4. Similar to (1), but with a three-byte field in the header tail, providing for over 16 million international standards.

5. Two bytes in the header tail indicate the standard identification number, the first byte indicating the sovereign state (see Section 5.1.2.3).

6. Similar to (5), except that four bytes are available (one for the sovereign state).

7. Similar to (5), except that eight bytes are available.

8. To be determined, or reserved for the distant future.

These first four options provide one-, two-, and three-byte standard-identification numbers to some designated international standards body or bodies. Two versions of the one-byte option are provided because otherwise too few combinations would be available. The remaining options provide one byte (or more) which identifies the sovereign state under whose authority the remaining standard-identification-number bytes the ID field have been assigned (1, 3, or 7 bytes remain). Note that receiving equipment would generally not distinguish between sovereign state identifiers and standard identification numbers; they would be treated together only as a single merged number that was known or unknown.

5.2.4 Header Tail

5.2.4.1 Header Tail Organization

The header tail contains one field for indicating the block length in integer format, one field for the standard identification number, and optional fields for error protection, all having been discussed above.

5.2.4.2 Header Tail: Standard-ID Organization

The Standard ID comprises two parts: the sovereign state identification number, and the standard identification number, described in Sections 5.2.4.3 and 5.2.4.4, respectively.

5.2.4.3 Header Tail: Sovereign State Identification

Eight bits is sufficient to designate the authorizing sovereign state, even if the number of sovereign states exceeds 256; the trick is to subdivide certain undesignated sovereign state identifiers by borrowing bits from the remaining standard identifier field. For example, by deferring to the United Nations the task of defining sovereignty, it should be an easy matter to assign all present U.N. members unique identification numbers in alphabetical order, and to assign the next numbers in order of membership admission to states not replacing member states who are already members. Once 224 member states exist, new United Nations members would share the last 32 numbers. Within these 32 sovereign state "condominium" identifiers, an additional three bits would be borrowed from the Standard Identifier field, permitting eight new member states per condominium, or 256-32+(32x8) = 480 possible states. Of these state designators, 32 would be assigned to existing standards bodies. In the unlikely event more states are created, still more bits can be borrowed, permitting unlimited growth.

5.2.4.4 Header Tail: Standard Identifier

The Standard Identifier would be selected by the indicated Sovereign State or International Standards Body using existing procedures. To the extent standards already have unique identification numbers, those same numbers could be used here. To the extent they do not, each standards body should map their standards into numbers, preferably a compact or consecutive set. Most new standards might be introduced under long numbers associated with minor standards bodies, while standards achieving wide acceptance could be renamed with short ones. Since bits can be borrowed from the Standard Identifier field almost indefinitely, the system would permit in certain cases (the longer header options) for every individual ever born to become a sovereign definer of standards under some state's authority (the eight-byte option provides each sovereign state with 70 million billion numbers). In the same way, a standards body could allocate numbers well spaced numerically so that their least significant bits could become "user-allocated bits", functioning in lieu of, or in addition to, a descriptor.

Note that the cost of this kind of flexibility, as well as the other sorts of flexibility described earlier, is essentially negligible given the extensible nature of this header architecture. That is, the price of using long headers is paid principally by those who need them, while most users will prefer and use the shorter options.

5.2.5 Header Organization Illustrations

These options can be represented pictorially. The sixteen-bit composition of the header key consists of three parts: the four bits in the length-type LT field (l), the four bits in the ID field (i), and the eight core error protection bits (p); these protect only the key. The header key always consists of:

llllliiiipppppppp

If we designate this two-byte key by the symbol "K", and the bytes representing the block length field by "L", the bytes representing the standard identification number by "S", and the header parity bytes by "P", then for LT options 1 and 2 we have only K (two bytes). For LT options 3, 4, and 5, we have only KSS (four bytes), KSSSP (six bytes), and KSSSS (six bytes), respectively. In this case of extremely short blocks, the standard identification numbers SSS and SSSS, combined with the 4 bits in the ID field, would generally convey messages in their own right, and could also be used for a variety of transactional and transport functions. LT option 7 would permit the following possibilities: KLS, KLSS, KLSSS, KLSSSS and KLSSSSSSSS, where the number of bytes allocated to the Standard Identification number by S...S is specified by the three bits in the ID field (see 5.1.1.3). One likely option for HDTV might be LT option 12, combined with ID option 1, represented as KLLLLSPP and comprising nine bytes. The standard number contained in field S would be an international standard. If such an international standard were devised to have block lengths no larger than 64KB, and if one-bit error correction were adequate, then a six-byte HDTV option is available: KLLSP. Such short 5 or 6-byte header options could often be employed for non-video information. For example, KLSS (5 bytes) would accommodate 64,000 possible international standards employing data up to 256 bytes per block.

5.2.6 Header Architecture: Equipment Implications

Equipment interpreting such headers can be particularly simple. Since it normally would be reading a stream of blocks, and should know when a block begins, it could simply jump to one of the 64K words specified by the 16-bit header key, these words indicating the location of the bytes specifying the block length. An example of an algorithm to perform these header decoding functions appears in Section 5.2.8, and an illustrative program coded in C appears in Appendix B. Simpler equipment might simply look at the first four bits of the LT field, from which the same information can be deduced in an error-free environment. Equipment of intermediate complexity can use the key error protection bits to an intermediate degree. The ID bits immediately indicate the field where the Standard Identifier is located, and equipment should compare this to a list of standards it is prepared to process. After any indicated processing, the equipment moves directly to the next header at the specified number of bytes along the data stream.

Should synchronization be lost, it could readily be recovered in most cases of interest. For example, in the nine-byte HDTV header example above, these nine bytes would probably never change within a single broadcast program. HDTV broadcast receivers would simply scan the data stream for that particular nine-byte sequence, which could be used as a traditional synchronization block. This would work even if multiple header types were being interleaved.

Accidental synchronizations (very rare) in the data block would be recognizable because the indicated false block length would probably not lead to a valid header.

The most direct use of these header/descriptors would be as a series of bytes at the start of each data block, where the data blocks are then concatenated in a comma-less string of bits. Other uses could also be made, however. For example, a particular transport scheme (e.g. over-the-air HDTV) might package data blocks in discontinuous but fixed form within a larger data stream, this stream being characterized by the imbedded header/descriptors. A header/descriptor could also characterize (say) each frame of an HDTV sequence, and the same header/descriptor technique could also be used within each frame to communicate details about its internal structure. Such nesting of header/descriptors does not impair synchronization much, although an initial false synchronization could occur within a larger block; this would be quickly detected if the presumed block ended without a valid header following it. The search for a valid header could then resume.

5.2.7 Header Architecture: Transport Functionality

If one wishes to incorporate synchronization augmentation, extra error protection, block prioritization information, or cryptography for the header or descriptor, it must either be defined at the outset in the header/descriptor definitions, or its incorporation becomes standard-specific--but how could we know the standard if we were not synchronized or error-free, etc.?

For example, in high-error-rate environments users may wish to provide their header/descriptors with more synchronization information or error protection than is available in the basic definitions. Fortunately, much additional transport-related information can be conveyed efficiently through repetition of short blocks. One simple approach is to insert brief bursts of short 2-byte headers into the data stream, where the synchronization powers of these 2-byte header blocks are cumulative; the greater the anticipated channel noise, the longer these bursts should be. Although the software of some receivers may not be sophisticated enough to synchronize such bursts well, this approach is standard independent, and so such software could be designed today. Such a burst inserted anywhere in a sequence of blocks permits synchronization of the entire stream. Since two-byte headers can correct two bits in 16, synchronization bursts fail only when the bit error-rate continually exceeds ~0.2. In still higher noise environments where it is known such two-byte bursts might be employed, the receiver can either autocorrelate the signal or correlate it with potential 16-bit synchronization words; this could provide synchronization for nearly any bit error rate, provided the burst length was sufficiently long.

Similar issues arise when extra error protection for the header/descriptor is desired. If the nature of such protection is defined only in a data or standard-ID area which can not be read unless error protection is employed, the data is generally inaccessible. One option is for the user to generate, as above, bursts of short blocks that convey primarily the identity of the desired error protection scheme employed in the longer data blocks. Such standards could be defined so that they are presumed operative for all following data until "turned off" by another command. Alternatively one of the short blocks (say 2 bytes) could be employed to indicate that error protection, cryptography, or other such schemes were to be conveyed in a "turn-on, turn-off" fashion; a separate short block could be used to convey the opposite message. Such a repetitious series of short blocks can also be well protected and synchronized in essentially any reasonable error environment (BER <0.2) using the existing defined header/descriptor options. In this case the user would need to know only the definition of the chosen error-protection ID number. Such error-protection schemes and protocols can be defined in simple ways over the years.

Yet another similar problem involves the possible incorporation of block priority information. For example, if some data processing, transmission, storage, or display step can not handle all the data, the originator of that data might wish to tell the user which data is more expendable. Yet the user might wish to do only minimal decoding to determine priority. Although such information would then be descriptor-standard specific, one or more descriptor standard ID's could be defined in such a way as to convey relative block priority, say on a scale from one to ten. Such information could also be imbedded in the data itself, but this would generally require the user to do more decoding before discarding any block. Although confusion could arise because multiple priority-labeling descriptor schemes might arise, they would all have standard numbers which, in principle, could be accessed. Section 5.4 describes another approach, which can be used in parallel.

Although this discussion has not been exhaustive, it does suggest that the proposed header/descriptor definition has great flexibility for handling a variety of problems faced when it must supplement or provide transport-layer or other OSI layer functions.

5.2.8 Compact-Header Decoding Algorithm

For simplicity here, we assume the block has been synchronized and that the header key has been error corrected, perhaps by using a 16-bit dispatch table. We also assume the equipment is provided with a list of standard ID's which it knows how to interpret, as well as a much shorter list of ID length fields corresponding to these standards (possible values of K, defined below).

1. Dispatch on the first byte (256 options); return with 4 integers:
   I = length of length field (0+, 0-, 1, 2, 4, or 6 bytes)
   J = block length B bytes if I = 0 (2, 4, or 6 bytes)
   K = length of ID field in tail (1, 2, 4, 8, or 32 bytes)
   L = presence of descriptor (0 or 1)
2. If I = 0, set block length B = J
3. If I = 0-, dispatch (Table 1) on M, bits 5-8 of the header
   Interpret resulting message and go to 12, or skip directly to 12 if message unknown.
4. If I = 0+, dispatch (Table 2) on bits 5-8 of header.
   Interpret resulting message and go to 12, or skip directly to 12 if message unknown.
5. If I ≠ 0, read I bytes (yielding block length B), starting at bit 17 of header.

6. If ID field length K not on list of known ID field lengths, go to 12.
7. Read ID field of length K bytes, starting at bit 17 + 8I.
8. If ID is not on list of known ID's, then go to 12.
9. If L = 1, read descriptor length D (part of descriptor decoding algorithm, not described here).
10. If L = 0, then D = 0.
11. Go to algorithm specified by ID and execute over a block of B bytes, starting at the end of the descriptor at I + K + D bytes.
12. Jump to end of block (B bytes long) and read next header.

5.3 Descriptor Specification

5.3.1 Introduction

Publicly readable descriptors may or may not be incorporated in any data block, as indicated by one of the bits in the header key. They would convey auxiliary information concerning the nature of the associated data, such as authorship, distributorship, ownership, intellectual property restrictions, sampling patterns, filtering employed, color, nonlinearities, etc. This information would generally be in the form of identification numbers assigned by standards bodies, although options for conveying text, programs, or other data would exist. Like the header, descriptors would indicate their length so that equipment could skip past if desired, and they would have optional provisions for error protection. An efficient ASN.1 equivalent for the descriptor definition proposed below could also be developed.

5.3.2 Descriptor Architecture

The descriptor is divided into three parts: a 2-byte "key", a "core" ranging from 0-8 bytes, and the "tail" of length defined by the core. The key unlocks the core, which defines the length of the descriptor, an indication of the nature of the contents of the descriptor, and the nature of any optional error protection for the core. The tail consists of a series of descriptor identification numbers, similar in concept to the standard identification numbers provided in the header. For each camera type, non-linear luminance mapping, movie producer, filtering algorithm, royalty payment procedure, etc., there could be a separate descriptor number assigned or registered by appropriate standards bodies, indicated in a manner also similar to that of the header. To properly convey this list of descriptor identification numbers, the tail also contains fields giving the number of such descriptor standards, the length and type of each such standard number, and the nature of any optional error protection employed. The tail also supports delivery of fields of text in any of a large number of languages, such as English or Portuguese, as well as computer languages such C, Postscript, etc. Such software elements would permit the decoding procedures to be defined explicitly, if desired.

A potential area for future improvement is development of a more universal subset of descriptor elements for widespread usage. It would include parameters such as resolution, raster definition, bit packing, etc.

5.3.3 Descriptor Key Definition

5.3.3.1 Descriptor Key Architecture

The descriptor key consists of three parts: 1) a 4-bit "type" field which characterizes the contents of the descriptor, 2) a 4-bit "length type" field which indicates the format of the descriptor length field, and 3) an 8-bit "protection" field for the key.

5.3.3.2 Descriptor Key: 4-Bit "Type" T Field

The purpose of the T field is to indicate: 1) whether or not this descriptor contains a public index (1 bit), 2) whether the descriptor length field in the core (if any) is 2 or 4 bytes long in integer format (1 bit), and 3) which of four error-protection options are being employed (2 bits). The four descriptor error-protection options are: 1) no protection, 2) protected core plus unprotected tail, 3) both core and tail protected, and 4) both core and tail doubly protected.

5.3.3.3 Descriptor Key: 4-Bit "Length-Type" DLT Field

This field contains the length of the descriptor after the core, in bytes, unless its contents are "zero, zero, zero, zero" (for descriptor tails longer than 16 bytes), in which case the length is given by the core in a field which is either 2 or 4 bytes long, as specified in the T field (see 5.3.3.2).

5.2.3.4 Descriptor Key: 8-Bit "Protection" P Field

The function of the P field is identical to that of the 8-bit error-protection field of the header key; it protects the 2-byte descriptor key only.

5.3.4 Descriptor Core Definition

5.3.4.1 Descriptor Core Architecture

The descriptor core consists of three parts: 1) a field defining the descriptor length (0-4 bytes), 2) a field indicating the nature of the contents of the descriptor (0 or 2 bytes), and 3) an optional protection field for the core only (0, 1, or 2 bytes). The total length of the descriptor core thus ranges between 0 and 8 bytes.

5.3.4.2 Descriptor Length Field

This field is of length zero if the descriptor length specification has been preempted by the length field in the descriptor key; otherwise it is either 2 or 4 bytes long in integer format, as determined by one of the bits in the T field of the descriptor key (see 5.3.3.2). The length of the descriptor field as presented in the core is defined as including all the bytes in the descriptor, including those in the key, core, and tail.

5.3.4.3 Descriptor Core Contents Index

This field contains either 0 or 2 bytes, as indicated by one of the bits in the descriptor key T field (see 5.3.3.2). In its 2-byte form it indicates whether or not the following descriptor contains information concerning any of sixteen categories of information about the data stream. Among others, these categories of information include synchronization reinforcement, error protection data, encryption keys, packet priorities, authorship, distributorship, time or date of any event, ownership, intellectual property restrictions, sampling patterns, filtering history, color, nonlinear mappings employed, etc. The last bit of the index is zero if this descriptor is the same as the previous one associated with the same header ID. The purpose of this contents index is to spare equipment the burden of decoding descriptors when their contents may be of no interest. This is particularly so when a long sequence of descriptor elements is repeated periodically to aid certain users having only segments of the data stream available to them. Users of longer segments could therefore ignore such data more readily.

5.3.4.4 Descriptor Core Parity Protection

This field would contain 0, 1, or 2 bytes, as indicated by two of the bits in the descriptor key T field (see 5.3.3.2). These bytes would protect the core only, using codes similar to those employed in the header.

5.3.5 Descriptor Tail Definition

5.3.5.1 Descriptor Tail Architecture

The descriptor tail consists of four fields:

1) the element-number field, which indicates the number of independent descriptor identification numbers contained in this descriptor; its length ranges from 4 bits to a maximum of 2 bytes;

2) the descriptor element length-type field, which specifies the lengths of each of the descriptor identification numbers contained in the following field, together with their respective types; these types include identification number types similar to those employed for indicating standard numbers in the header, as well as supporting transmittance of text and computer programs; its length is typically 2-4 bits per descriptor identification number;

3) the descriptor identification number field, which consists typically of one or more descriptor standard numbers, each in a 1-6 byte format or appearing as a sequence of text or code; the total length of this field approximates several bytes per descriptor element, or substantially more if text or code is incorporated; and

4) the protection field, as defined jointly by the protection option indicated in two of the T bits (see 5.3.3.2) in the descriptor key combined with the indicated descriptor length; longer descriptor lengths would require more bytes of protection for any indicated level of protection.

5.3.5.2 Descriptor Tail Element-Number Field

This field consists of one, two, three, or four 4-bit words indicating the number of descriptor elements contained within this descriptor. Each 4-bit word contains 3 bits indicating the number of elements, and 1 bit indicating whether or not an additional 4-bit word is appended, up to a maximum of four words total. Thus one 4-bit word will suffice for 0-7 descriptor types, which normally should be sufficient. Two concatenated 4-bit words offer up to 2 exp 6 = 64 possible elements. Three words can accommodate up to 512 elements, while use of all four words (2 bytes) can accommodate more 8,000 elements, which should be sufficient and is the maximum number per header allowed under this protocol.

5.3.5.3 Descriptor Tail Element Length-Type Field

This field consists of a series of extensible 2-bit words, one sequence of such words applying to each descriptor element. In most cases a single 2-bit word would suffice; the options here are that a 1-, 2-, or 3-byte field is reserved for the associated descriptor identification number; the fourth option is that two 2-bit words are being employed. If the second 2-bit word is employed, the associated options are that 4, 5, or 6 bytes are being employed to indicate the associated descriptor identification number; this accommodates up to 10 exp 12 possible identification numbers, which should be adequate. The fourth option available for the second 2-bit word indicates that an additional 4-bit word is to be interpreted. This 4-bit word offers 16 additional options, the first of which is that following the 4-bit word, a 1-byte word specifies the length of the descriptor identification field.

The remaining 15 options are of similar form, but indicate that types of descriptor data are being employed other than the standard identification number type. For example, type 2 would indicate that ASCII text was being employed in a language indicated by the first character of the text stream; thus 256 possible languages can be used. Types 3-16 would indicate which of several possible computer languages or image description formats were being employed, such as C, Postscript, etc. If it is felt that the 15 possible languages available under the 4-bit extension option in the element length-type field is inadequate, then additional 4-bit fields could be appended by using an extension bit, or by using 1 bit in each 4-bit field to indicate additional 4-bit fields are appended and can be interpreted as were the series of descriptor tail element-number 4-bit words. (Alternatively, the 4-bit word could be reduced to 1 or 2, with the understanding that the language is specified by the first following 2 bytes.) Definition of these options is left to the next step in the SMPTE standards definition process.

5.3.5.4 Descriptor Tail Standard Identification Numbers

If one or two 2-bit words have been previously employed to indicate the length of the descriptor identification number, then 1-256 bytes may be employed for the ID number itself. The formats for each of these options are indicated below.

1 byte	International standard established by single designated authority

2 bytes	16-bit standard number designated by authorized international standards body (bodies)
3-256 bytes	1 byte indicating the sovereign state, and the remainder (2-255 bytes) being available for the standards number

The sovereign state and standards numbers would be designated using procedures similar to that specified in the header. Note that the longer standards numbers permit subdesignations under the sovereign state indicator for subsidiary standards bodies, including individual corporations, institutions, and even individuals. The longer fields also permit use of user-defined bits which can be assigned at execution, thus providing a data field. Such data fields could be used for conveying dynamically changing information such as average luminance, audio gain, etc.

5.3.6 Descriptor Tail Error Protection

This field could be concentrated at the end of the descriptor or distributed throughout to simplify processing. The number of bytes and protocol employed for this purpose would be determined uniquely by the 2 bits in the descriptor key T field and the descriptor length, as specified in the descriptor core descriptor length field or in the descriptor key length field. Definition of these protection strategies might parallel those employed in the header and remain to be defined more fully.

5.4 Transport Header/Descriptors

5.4.1 Motivation and Objectives

Section 5.2.7 discusses several reasons why providing error protection, synchronization reinforcement, packet priority, and higher level encryption to header/descriptors could pose problems for interpretive hardware if the data is excessively noisy. Although the solutions suggested there will accommodate most error environments, still more serious situations can be handled using a transport header/descriptor block such as described here. Such a transport block has the additional advantage that if insufficiently protected data is moving into a more hostile transport environment, additional protection can be incorporated in the transport block without having to redefine the input blocks. Similarly, such transport blocks can be removed without penalty when moving into more error-free or otherwise benign environments.

The objective here is to suggest how the architecture of such transport blocks are consistent with the header/descriptor definitions presented above, but not to define all the details. Thus establishment of the header/descriptor standard can proceed without waiting for all details of the transport block to be resolved. It would be useful to resolve such details, however, before users of the standard adopt inferior methods of addressing the same transport problems.

The principal motivation for defining transport blocks is to avoid proliferation of standard-specific alternatives for addressing such transport problems. Such proliferation could increase the cost of decoding equipment which would have to accommodate all these possibilities. In a high-error environment, executing the multiple search strategies necessary when synchronization is lost or heavy errors exist, could become prohibitive. For this reason it is important to have only a few standard options for certain aspects of the transport block. Defining an efficient small set is beyond the scope of the present effort, and would take considerable study. Therefore it is reasonable to assume that this study would be completed after any initial header/descriptor standard is specified. One illustrative candidate for such a set appears here in Appendix A; it is intended only to initiate discussion of these issues.

5.4.2 Architecture of Transport Blocks

Transport header/descriptor blocks would consist of a single header/descriptor, where the header would specify an international standard ID indicating which type of transport block was involved; only a few such types would ever be defined. The descriptor of the transport block would convey up to eight different elements:

1) Descriptor Table of Contents (standard format defined earlier).

2) Synchronization reinforcement bits, not error protected.

3) Error protection bits for the transport header and its attached following header/descriptor.

4) Encryption key for deciphering the descriptor, if any, in the following block.

5) Block priority; determined by data originator, indicating relative priority of the following block concerning interpretation or transmission in cases where inadequate capacity is available. Authorization keys may also be needed to verify priority in certain cases. Price bidding could be supported here too.

6) Authorizations and fee mechanisms for alteration or use of data.

7) Block sequence numbering and timing-reconstruction information.

8) Padding to yield one of the very few allowed lengths for the transport block corresponding to the international header standard ID.

In addition to these elements there would also be the traditional field in the descriptor defining its length, although interpretation of this length would be unnecessary because the transport block length is specified by the header, and there is no data payload following the descriptor.

The six main descriptor elements would convey information using traditional descriptor standard numbers, where long numbers accommodating an adequate number of user bits could be employed. Defining these descriptor standard numbers is the task which can and probably must be postponed until the technical tradeoffs associated with different choices are better understood.

Thus standards for headers and descriptors can and should be adopted in advance of these descriptor definitions for transport blocks.

5.4.3 Decoding Issues for Transport Headers

Decoding such a transport header in a high-error environment would be relatively straightforward. First, if synchronization had been lost, synchronization would be established. Initially this might be done assuming a low-error environment. If the environment is noisy, then each of a few possible synchronization blocks would be sought, including periodic repetition of legal header keys, assuming that key bursts might have been employed for this purpose. Because all possible synchronization words might be sought, it is important to have only a few legal ones if they are many bytes long. Because the synch reinforcement bits could be only in a small number of positions relative to the beginning of the transport block, each such position could be tested for consistency with the associated error protection bits, which are also located in a small number of possible locations. Confirmation of synchronization follows if legal headers come immediately after the indicated end of any block. Once the transport block is synchronized and error corrected, the remaining descriptor fields containing any encryption key for the descriptor in the following block, or any packet priority information can be deciphered.

One principal new constraint should be imposed by the standard on manufacturers of equipment handling this header/descriptor standard. If transport blocks are to be useful, such equipment must never insert data between a transport block and the following block to which it applies. Transport systems should try not to scramble the sequence of data blocks in any event, but if a transport block should accidentally be prepended to the wrong data block, the packet priority, encryption key for the data block descriptor, and the error protection for the header/descriptor could be inappropriate, resulting in a scrambling of the interpreted header/descriptor. Such scrambling would also typically cause local loss of synchronization, particularly in high-error environments. Since transport blocks would normally be quite brief compared to typical data blocks, such a constraint should not be difficult to satisfy. Such transport blocks could also be added or subtracted at will by a given transport layer, however, provided they are appropriate to the blocks which they precede.

If a data stream is entered in the middle of a transport block, then confusion might result. To protect against this unlikely possibility, equipment might choose to wait until the second valid header is intercepted before commencing decoding.

6. ABSTRACT SYNTAX NOTATION 1 (ASN.1) HEADER/DESCRIPTOR ARCHITECTURE

6.1 Background

Abstract Syntax Notation 1 (ASN.1) is an existing ISO/CCITT standard in common use within the computer and telecommunications industries. Within the ASN.1 framework, it is straightforward to define a SMPTE header/descriptor that meets the objectives described in Sections 1-5 above, and it would leverage existing tools, expertise, and administrative structures.

ASN.1 is derived from earlier work at Xerox PARC on Courier (late 1970s). An early version of the notation (c. 1984) was used in the first draft of the CCITT X.400 series of recommendations on message handling systems (i.e., electronic mail). ISO and CCITT then jointly developed ASN.1 for use within the OSI presentation layer (c. 1988).

ASN.1 is now widely used in a range of standards activities, including the CCITT X.500 directory service and both the OSI and Internet network management systems. Over the years, a collection of software tools and utilities to support ASN.1 has been (and is being) developed.

6.2 Concepts

ASN.1 is an extensible notation for describing data that is to be exchanged by transmission or storage. It is much like a programming language, such as C and Pascal. There are several simple types, such as integer, real, and octet string (i.e., byte string), and constructor types that can be used to build arbitrarily complex data structures, including hierarchical representations (e.g., packet within packet).

An ASN.1 header can be thought of as an envelope that contains, for example, a single video frame. ASN.1 supports the notion of embedding, which allows one or more data structures to be contained within another. Thus, a sequence of frames can be embedded within an outer header (or envelope) that labels a program segment. This can be taken to coarser granularity, e.g., shots, scenes, programs, etc. Similarly, it can be taken to finer granularity to embed audio tracks, closed captioning, descriptors, etc. within individual frames.

A key feature in ASN.1 is the separation of how the data is described (Abstract Syntax) and how data is encoded (Basic Encoding Rules, "BER"). Data structures are described in a human-readable syntax and automatically translated into the bits and bytes for transfer. When a new data structure (or type) is defined, its representation is automatically generated. Furthermore, deployed ASN.1 compliant systems will be able to interpret new structures without hardware modification.

The following summary description of ASN.1 presents only enough detail to motivate its use for the specific needs of a header/descriptor. For formal definition of ASN.1 refer to ISO 8824/8825 and/or CCITT X.208/209. A more accessible description can be found in: Marshall T. Rose, The Open Book: A Practical Perspective on OSI, Prentice Hall, 1990.

6.2 Basic Encoding Rules (BER)

All ASN.1 types, whether a simple type or a structured type, can be encoded using the same basic format of three fields:

```
[ tag ] [ length ] [ value ]
```

The three fields together make up a data item. Each field is variable in size to accommodate arbitrarily complex substructures and encodings. A simple type, such as an integer, requires only a few bytes. A structured type, such as a long byte string, can be megabytes, gigabytes, or larger as necessary to contain the payload data value. The basic format is inherently self identifying and extensible.

A data stream is a sequence of items each of which can be structured or nested. Thereby, one can define arbitrarily structured data for both header and descriptor, including nested packet-within-packet structures.

Tag Field. The tag field specifies the type of the item value. Several simple and structured types (integers, character strings, etc.) are universally defined in the ASN.1 standard, and are recognized in all compliant environments. Also, one can define types that exist within the specific environment of an application or communications context. A tag is principally encoded as a single byte, but can be extended. In the BER, the tag is encoded as:

```
  <2>   <1>    <5>          -- bits per field
[ class | p | tag number ]
```

The tag field allows a receiver to "parse" the incoming data stream, selecting those components/types in which it is interested and bypassing others.

The EXTERNAL type is of particular significance. In essence, it is a universal header for the data that it encapsulates. The EXTERNAL type is described further below. Other types are relevant to use in a descriptor.

Length Field. The length field indicates the size of the value. It is an integer of one or more bytes that specifies the number of bytes in the value field. In the BER, the short form of length is encoded as a single byte, and can indicate lengths of 0..126 bytes of value. In extended form, the first byte specifies the number of bytes of length. Length is encoded as:

```
short form    : 1 byte  : [ 0bbbbbbb ] : lengths of 0..126 bytes

extended form : n bytes : [ 1nnnnnnn ] [ bbbbbbbb ] [ bbbbbbbb ] ...n
```

Only the number of bytes needed to specify length are used. Thus, the length field is compact. The extensibility of the length field permits a maximum length field that is 126 bytes to specify a length of ~2^1008 or ~10^303.

Note that a common length specification is used regardless of the associated data item. Accordingly, there is no need to invent custom length encoding schemes for each new data item.

Value Field. The value field is the value in the type specified by the tag. It is